

# Parametrizing Program Analysis by Lifting to Cardinal Power Domains

Lunjin Lu

Oakland University, MI 48309, USA

**Abstract.** A parametric analysis is an analysis whose input and output are parametrized with a number of parameters which can be instantiated to abstract properties after analysis is completed. This paper proposes to use Cousot and Cousot's Cardinal power domain to capture functional dependencies of analysis output on its input and obtain a parametric analysis by parametrizing a non-parametric base analysis. We illustrate the method by parametrizing a *Pos* based groundness analysis of logic programs to a parametric groundness analysis. In addition, a prototype implementation shows that generality of the parametric groundness analysis comes with a negligible extra cost.

## 1 Introduction

A program analysis is to infer information from programs. Let  $P$  be a program,  $\mathcal{I}$  express input information before analysis, and  $\mathcal{O}$  express output information inferred from  $P$  and  $\mathcal{I}$ . We write  $\langle \mathcal{I}, P, \mathcal{O} \rangle$  to denote the analysis that infers  $\mathcal{O}$  from  $P$  and  $\mathcal{I}$ . A typical program analysis is non-parametric in the sense that the program need be analyzed separately for different input information. Note that program variables are not parameters for input information, though input information can be thought of as predicates over program variables. Take the generic sorting program  $sort(x, y)$  for instance, letting  $nat$  denote the set of natural numbers,  $int$  the set of integers, and  $list(\beta)$  the set of lists of elements from  $\beta$ , program analyses  $\langle x \in list(nat), sort(x, y), y \in list(nat) \rangle$  and  $\langle x \in list(int), sort(x, y), y \in list(int) \rangle$  are accomplished separately even if they are two instances of a parametric analysis  $\langle x \in list(\beta), sort(x, y), y \in list(\beta) \rangle$  where both input information and output information are parametrized. By assigning different values to  $\beta$  which serves as a place holder for information to be filled in after analysis,  $\langle x \in list(\beta), sort(x, y), y \in list(\beta) \rangle$  can be instantiated into many different non-parametric analyses such as  $\langle x \in list(nat), sort(x, y), y \in list(nat) \rangle$  and  $\langle x \in list(int), sort(x, y), y \in list(int) \rangle$ . Parametric program analyses infer more general results, which brings some benefits. Firstly, a sub-program or a library program need not be analyzed separately for its different uses, i.e., the result of a parametric analysis is re-usable. This has positive bearing on efficiency of analysis because output information for different uses of the same sub-program can be obtained by instantiation rather than by re-analysis. Secondly, parametric analyses are amenable to program modifications since changes to the program

does not necessitate re-analyses of the sub-program so long as the sub-program itself is not changed.

This paper addresses the issue of lifting a non-parametric analysis to a parametric analysis such that each instantiation of the result of running the parametric analysis is same as the result of running the non-parametric analysis with instantiated input information. If  $\langle \mathcal{J}(\beta), P, \mathcal{O}(\beta) \rangle$  is the result of the parametric analysis then  $\langle \mathcal{J}(\kappa), P, \mathcal{O}(\kappa) \rangle$  is the result of the non-parametric analysis for any possible value  $\kappa$  for  $\beta$ . Observe that both input  $\mathcal{J}(\beta)$  and output  $\mathcal{O}(\beta)$  of the parametric analysis are functions from the domain of values for parameters to the domain of input properties non-parametric analysis.

The contributions of the paper are as follows. Firstly, a systematic approach is presented for deriving a parametric analysis from a given non-parametric base analysis. This involves lifting the semantic domain for the base analysis to its Cardinal power with respect to the domain of parameter values and lifting the semantic function accordingly. Secondly, this approach is applied to a goal-dependent groundness analysis for logic programs using parameters to express groundness of variables in the top-level goal. The result is a parametric goal dependent groundness analysis. Thirdly, we present an encoding for the abstract properties and abstract operations for the parametric groundness analysis using positive propositional formulas.

The next section provides background knowledge on abstract interpretation and logic program analysis. Section 3 describes the approach to parametrizing program analyses and section 4 obtains the parametric goal-dependent groundness analysis for logic programs by applying the approach. Section 5 presents the encoding and section 6 some experimental results with a prototype implementation of the parametric groundness analysis. We discuss related work in section 7 and then conclude in section 8.

## 2 Preliminaries

### 2.1 Abstract Interpretation

A semantics of a program is given by an interpretation  $\langle (D, \sqsubseteq_D), f \rangle$  where  $(D, \sqsubseteq_D)$  is a complete lattice and  $f$  is a monotone function on  $(D, \sqsubseteq_D)$ . The semantics is defined as the least fixed point  $\text{lfp } f$  of  $f$ . The concrete semantics of the program is given by the concrete interpretation  $\langle (D, \sqsubseteq_D), f \rangle$  while an abstract semantics is given by an abstract interpretation  $\langle (D^\#, \sqsubseteq_{D^\#}), f^\# \rangle$ . The correspondence between the concrete and the abstract domains is formalized by a Galois connection  $(\alpha, \gamma)$  between  $(D, \sqsubseteq_D)$  and  $(D^\#, \sqsubseteq_{D^\#})$ . A Galois connection between  $D^\#$  and  $D$  is a pair of monotone functions  $\alpha : D \mapsto D^\#$  and  $\gamma : D^\# \mapsto D$  satisfying  $\forall c \in D. (c \sqsubseteq_D \gamma \circ \alpha(c))$  and  $\forall a \in D^\#. (\alpha \circ \gamma(a) \sqsubseteq_D a)$ . The function  $\alpha$  is called an abstraction function and the function  $\gamma$  a concretization function. A sufficient condition for  $\text{lfp } f^\#$  to be a safe abstraction of  $\text{lfp } f$  is  $\forall a \in D^\#. (\alpha \circ f \circ \gamma(a) \sqsubseteq_{D^\#} f^\#(a))$  or equivalently  $\forall a \in D^\#. (f \circ \gamma(a) \sqsubseteq_D \gamma \circ f^\#(a))$ , according to propositions 24 and 25 in [9]. In a compositional design of analysis, the concrete semantics is defined in terms of a group of semantic functions

$f_i : D_i \mapsto E_i$  and the abstract semantics is defined in terms of another group of semantic function  $f_i^\sharp : D_i^\sharp \mapsto E_i^\sharp$  such that each abstract semantic function  $f_i^\sharp$  simulates its corresponding concrete semantic function  $f_i$ . To prove the correctness of the abstract semantics with respect to the concrete semantics is reduced to proving the correctness of each abstract semantic function  $f_i^\sharp$  with respect to its corresponding concrete semantic function  $f_i$ . Let  $\gamma_{D_i^\sharp} : D_i^\sharp \mapsto D_i$  and  $\gamma_{E_i^\sharp} : E_i^\sharp \mapsto E_i$  be concretization functions. Then  $f_i^\sharp : D_i^\sharp \mapsto E_i^\sharp$  is correct with respect to  $f_i : D_i \mapsto E_i$  iff  $f_i(\gamma_{D_i^\sharp}(x^\sharp)) \sqsubseteq_{E_i} \gamma_{E_i^\sharp}(f_i^\sharp(x^\sharp))$  for each  $x^\sharp \in D_i^\sharp$ .

## 2.2 Logic Program Analysis

Assume a set of function symbols  $\Sigma$  and an infinite set of variables  $\mathcal{V}$ . Let  $V \subseteq \mathcal{V}$ . Then  $\text{Term}(\Sigma, V)$  denotes the set of all terms that can be constructed from  $\Sigma$  and  $V$ . Let  $\mathbf{V}(o)$  be the set of variables in a syntactic object  $o$ . A bold lower case letter denotes a sequence of different variables. When there is no ambiguity,  $\mathbf{V}(\mathbf{x})$  will be written as  $\mathbf{x}$ . The set of subsets of a set  $S$  is denoted by  $\wp(S)$  and the set of finite subsets of  $S$  by  $\wp_f(S)$ .

*Substitutions* A substitution  $\theta$  is a mapping from  $\mathcal{V}$  to  $\text{Term}(\Sigma, \mathcal{V})$  such that its domain  $\text{dom}(\theta) = \{x \mid x \neq \theta(x)\}$  is finite. A substitution  $\theta$  is idempotent iff  $\theta(\theta(x)) = \theta(x)$  for all  $x \in \mathcal{V}$ . The set of all idempotent substitutions is denoted  $\text{Subst}$ . A renaming substitution is a bijection from  $\mathcal{V}$  to  $\mathcal{V}$ . Let  $\sim_U$  be the equivalence relation defined  $\theta \sim_U \sigma$  iff there is a renaming substitution  $\rho$  such that  $\theta(x) = \rho(\sigma(x))$  for all  $x \in U$  [1]. Let  $[\theta]_U$  denote the equivalence class of  $\theta$  with respect to  $\sim_U$  and  $\text{Subst}_U$  the quotient set of  $\text{Subst}$  with respect to  $\sim_U$ . A substitution  $\theta'$  is a canonical representative of an equivalence class  $[\theta]_U$  iff  $\theta' \in [\theta]_U$  and  $\text{dom}(\theta') = U$  and  $U \cap \text{rng}(\theta') = \emptyset$  where  $\text{rng}(\theta') = \bigcup_{x \in \text{dom}(\theta')} \mathbf{V}(\theta'(x))$ . In any  $\theta'$  in  $[\theta]_U$ , bindings for variables outside  $U$  is irrelevant since  $\{x \mapsto \theta'(x) \mid x \in U\} \sim_U \theta'$ . Let  $\diamond \notin \text{Subst}_U$  for any  $U \subseteq \mathcal{V}$ .

*Operations on substitutions* An equational constraint is a finite set (conjunction) of equations of the form  $t_1 = t_2$  with  $t_i$  for  $i = 1, 2$  being terms. Define  $\text{mgu}(E)$  as the  $\sim_{\mathbf{V}(E)}$  equivalence class of most general unifiers for  $E$  if  $E$  is unifiable. Otherwise,  $\text{mgu}(E) = \diamond$ . One operation performed during program execution is to conjoin constraints represented by substitutions. The unification operation  $\odot : \text{Subst}_U \times \text{Subst}_V \mapsto \text{Subst}_{U \cup V} \cup \{\diamond\}$  is defined by  $[\theta_1]_U \odot [\theta_2]_V = \text{mgu}(\text{eq}(\theta'_1) \cup \text{eq}(\theta'_2))$  where  $\text{eq}(\theta) = \{x = \theta(x) \mid x \in \text{dom}(\theta)\}$  and  $\theta'_1$  and  $\theta'_2$  are respectively canonical representatives of  $[\theta_1]_U$  and  $[\theta_2]_V$  such that  $(U \cup \mathbf{V}(\theta'_1)) \cap (V \cup \mathbf{V}(\theta'_2)) \subseteq U \cap V$ . Another operation is projection  $\pi_X : \text{Subst}_U \mapsto \text{Subst}_{U \setminus X}$  for  $X \in \wp_f(\mathcal{V})$  defined as  $\pi_X([\theta]_U) = [\theta]_{U \setminus X}$ . The operator  $\pi_X$  hides variables in  $X$ . A third operation is renaming defined as follows. If  $\mathbf{x} \cap \mathbf{y} = \emptyset$  then  $\mathcal{R}_{\mathbf{x} \rightarrow \mathbf{y}}(\eta) = \pi_{\mathbf{x}}(\text{mgu}(\{\mathbf{x} = \mathbf{y}\}) \odot \eta)$ . Otherwise,  $\mathcal{R}_{\mathbf{x} \rightarrow \mathbf{y}}(\eta) = \mathcal{R}_{\mathbf{z} \rightarrow \mathbf{y}}(\mathcal{R}_{\mathbf{x} \rightarrow \mathbf{z}}(\eta))$  where  $\mathbf{z} \cap (\mathbf{x} \cup \mathbf{y} \cup \mathbf{V}(\eta)) = \emptyset$ . Note that  $\text{mgu}(\{\mathbf{x} = \mathbf{y}\}) \neq \diamond$  since  $\mathbf{x}$  and  $\mathbf{y}$  are sequences of different variables.  $\mathcal{R}_{\mathbf{x} \rightarrow \mathbf{y}}(\cdot)$  transforms an equational constraint on  $\mathbf{x}$  to one on  $\mathbf{y}$ .

*Concrete and Abstract Interpretations* The concrete semantics for a logic program analysis is usually defined in terms of several operations on the collecting domains  $\Pi_{U \in \mathcal{U}} \langle \wp(\text{Subst}_U), \subseteq \rangle$  where each  $U \in \mathcal{U}$  is the set of program variables of interest at a specific program point and  $\mathcal{U}$  represents the set of program points of interest. The concrete interpretation is

$$I = \langle \Pi_{U \in \mathcal{U}} \langle \wp(\text{Subst}_U), \subseteq, \cup \rangle, \odot^*, \mathcal{R}_{x \mapsto y}^*, \pi_X^* \rangle$$

where  $\cup$  is the set union and  $\odot^*, \mathcal{R}_{x \mapsto y}^*$  and  $\pi_X^*$  are set extensions of  $\odot, \mathcal{R}_{x \mapsto y}$  and  $\pi_X$  respectively. An analysis is obtained by designing an abstract interpretation

$$I^\# = \langle \Pi_{U \in \mathcal{U}} \langle \text{ASub}_U^\#, \sqsubseteq_U^\#, \sqcup_U^\# \rangle, \odot^\#, \mathcal{R}_{x \mapsto y}^\#, \pi_X^\# \rangle$$

such that  $\langle \text{ASub}_U^\#, \sqsubseteq_U^\#, \sqcup_U^\# \rangle$  is related to  $\langle \wp(\text{Subst}_U), \subseteq, \cup \rangle$  with a Galois connection  $\langle \wp(\text{Subst}_U), \alpha_U, \text{ASub}_U^\#, \gamma_U \rangle$  and  $\odot^\#, \mathcal{R}_{x \mapsto y}^\#$  and  $\pi_X^\#$  approximate correctly  $\odot^*, \mathcal{R}_{x \mapsto y}^*$  and  $\pi_X^*$  respectively.

### 3 Parametrizing Program Analyses

An analysis  $\langle \mathcal{J}, P, \mathcal{O} \rangle$  computes  $\mathcal{O}$  as the limit of an ultimately stationary sequence of iterates:  $\mathcal{J}, f(\mathcal{J}), \dots, f^{(\lambda)}(\mathcal{J}), \dots$  where  $f$  is a monotone semantic function. The iteration of  $f$  is defined

$$\begin{aligned} f^{(0)} &= id \\ f^{(\lambda+1)} &= f \circ f^{(\lambda)} \\ f^{(\lambda)} &= \sqcup_{\beta < \lambda} f^{(\beta)} \text{ when } \lambda \text{ is a limit ordinal} \end{aligned}$$

The limit is denoted  $\text{lfp}_{\mathcal{J}} f$ . As an example, consider the forward collecting semantics which characterizes the set of the program states  $\mathcal{O}$  that can be reached from a set of initial states  $\mathcal{J}$ . According to proposition 33 in [9],  $\mathcal{O} = \text{lfp}_{\emptyset} F[P]$  where  $F[P](X) = \mathcal{J} \cup \text{post}[\xrightarrow{P}]X$  and  $\xrightarrow{P}$  is the transition relation between program states and  $\text{post}[t]S = \{s' \mid \exists s \in S. \langle s, s' \rangle \in t\}$ . It is easy to verify that  $\mathcal{O} = \text{lfp}_{\mathcal{J}} \lambda X. (X \cup \text{post}[\xrightarrow{P}]X)$ .

#### 3.1 Lifting Semantic Domains to Cardinal Power Domains

The cardinal power  $L_1 \xrightarrow{m} L_2$  with base  $L_2$  and exponent  $L_1$  consists of all monotone functions from  $L_1$  to  $L_2$ . We parametrize a base analysis by lifting both the concrete and the abstract domains to cardinal powers.

**Proposition 1.** [10] *Let  $\langle L_1, \alpha_1, L_1^\#, \gamma_1 \rangle$  and  $\langle L_2, \alpha_2, L_2^\#, \gamma_2 \rangle$  be Galois connections. Then  $\langle L_1 \xrightarrow{m} L_2, \alpha, L_1^\# \xrightarrow{m} L_2^\#, \gamma \rangle$  is a Galois connection where  $\alpha = \lambda \phi. \alpha_2 \circ \phi \circ \gamma_1$  and  $\gamma = \lambda \psi. \gamma_2 \circ \psi \circ \alpha_1$ .*

*Proof.* For any  $\phi \in L_1 \xrightarrow{m} L_2$ ,  $\phi \sqsubseteq \gamma_2 \circ \alpha_2 \circ \phi \circ \gamma_1 \circ \alpha_1 \sqsubseteq \gamma \circ \alpha(\phi)$  since  $\gamma_i \circ \alpha_i$  are extensive for  $i = 1, 2$ . For any  $\psi \in L_1^\# \xrightarrow{m} L_2^\#$ ,  $\alpha \circ \gamma(\psi) = \alpha_2 \circ \gamma_2 \circ \psi \circ \alpha_1 \circ \gamma_1 \sqsubseteq \psi$  since  $\alpha_i \circ \gamma_i$  are reductive for  $i = 1, 2$ . Since  $\gamma \circ \alpha$  is extensive and  $\alpha \circ \gamma$  is reductive,  $\langle L_1 \xrightarrow{m} L_2, \alpha, L_1^\# \xrightarrow{m} L_2^\#, \gamma \rangle$  is a Galois connection.  $\square$

### 3.2 Lifting Semantic Functions

The domain of an interpretation is often formed from a number of primitive domains and the semantic function from a number of primitive functions between primitive domains. We now define a family of operators  $\star_L$  that lift a monotone function  $f : D \xrightarrow{m} E$  to a monotone function  $\star_L f : (L \xrightarrow{m} D) \xrightarrow{m} (L \xrightarrow{m} E)$ .

**Definition 1.** Let  $f : D \xrightarrow{m} E$ . Define  $\star_L f : (L \xrightarrow{m} D) \xrightarrow{m} (L \xrightarrow{m} E)$  as

$$\star_L f = \lambda \phi. f \circ \phi$$

The following theorem shows that lifting of the semantic function of an interpretation can be accomplished by lifting individual primitive semantic functions.

**Theorem 1.** For any  $L$ ,

1.  $\star_L(f_2 \circ f_1) = (\star_L f_2) \circ (\star_L f_1)$  for any  $f_1 : D \xrightarrow{m} E$  and  $f_2 : E \xrightarrow{m} F$ ,
2.  $\star_L \langle f_1, f_2 \rangle = \langle \star_L f_1, \star_L f_2 \rangle$  for any  $f_1 : D \xrightarrow{m} E$  and  $f_2 : D \xrightarrow{m} F$ ,
3.  $\star_L \text{proj}_i(\langle \phi_1, \phi_2 \rangle) = \phi_i$  for  $i = 1, 2$ ,  $\phi_1 : L \xrightarrow{m} D_1$  and  $\phi_2 : L \xrightarrow{m} D_2$  where  $\text{proj}_i(\langle c_1, c_2 \rangle) = c_i$  for  $i = 1, 2$ .

*Proof.* Consider item (1) first. For any  $\phi \in L \xrightarrow{m} D$ ,  $\star_L(f_2 \circ f_1)(\phi) = f_2 \circ f_1 \circ \phi = f_2 \circ (\star_L f_1)(\phi) = (\star_L f_2)((\star_L f_1)(\phi)) = (\star_L f_2) \circ (\star_L f_1)(\phi)$ .

Now consider item (2). For any  $\phi \in L \xrightarrow{m} D$ ,  $\star_L \langle f_1, f_2 \rangle(\phi) = \langle f_1, f_2 \rangle \circ \phi = \langle f_1 \circ \phi, f_2 \circ \phi \rangle = \langle \lambda \psi_1. f_1 \circ \psi_1, \lambda \psi_2. f_2 \circ \psi_2 \rangle(\phi) = \langle \star_L f_1, \star_L f_2 \rangle(\phi)$ .

Item (3) follows from definition of  $\pi_i$  and  $\star_L$ .  $\square$

Let  $\langle L_2, \alpha_2, L_2^\sharp, \gamma_2 \rangle$  be a Galois connection and  $f : L_2 \xrightarrow{m} L_2$  and  $f^\sharp : L_2^\sharp \xrightarrow{m} L_2^\sharp$  the concrete and abstract semantic functions. The concrete and abstract domains  $L_2$  and  $L_2^\sharp$  can be parametrized by  $L_1$  and  $L_1^\sharp$  which are related to each other by a Galois connection  $\langle L_1, \alpha_1, L_1^\sharp, \gamma_1 \rangle$ . The following theorem says that  $\star_{L_1^\sharp} f^\sharp$  approximates  $\star_{L_1} f$  if  $f^\sharp$  approximates  $f$ . Furthermore, if  $f^\sharp$  is the best approximation of  $f$  and  $\langle L_1, \alpha_1, L_1^\sharp, \gamma_1 \rangle$  is a Galois insertion then  $\star_{L_1^\sharp} f^\sharp$  is the best approximation of  $\star_{L_1} f$ .

**Theorem 2.** Let  $\langle L_1, \alpha_1, L_1^\sharp, \gamma_1 \rangle$  and  $\langle L_2, \alpha_2, L_2^\sharp, \gamma_2 \rangle$  be Galois connections,  $f : L_2 \xrightarrow{m} L_2$  and  $f^\sharp : L_2^\sharp \xrightarrow{m} L_2^\sharp$ . Let  $\alpha$  and  $\gamma$  be defined as in Proposition 1. Then

1. If  $\alpha_2 \circ f \circ \gamma_2 \sqsubseteq f^\sharp$ ,  $\alpha \circ (\star_{L_1} f) \circ \gamma \sqsubseteq \star_{L_1^\sharp} f^\sharp$ .
2. If  $\alpha_2 \circ f \circ \gamma_2 = f^\sharp$  and  $\langle L_1, \alpha_1, L_1^\sharp, \gamma_1 \rangle$  is a Galois insertion,  $\alpha \circ (\star_{L_1} f) \circ \gamma = \star_{L_1^\sharp} f^\sharp$ .

*Proof.* Consider (1) first. Let  $\phi$  be an arbitrary member of  $L_1^\# \xrightarrow{m} L_2^\#$ .

$$\begin{aligned}
(\alpha \circ (\star_{L_1} f) \circ \gamma)(\phi) &= \alpha((\star_{L_1} f)(\gamma(\phi))) && \text{by def. of } \gamma \\
&= \alpha((\star_{L_1} f)(\gamma_2 \circ \phi \circ \alpha_1)) && \text{by def. of } \star_{L_1} f \\
&= \alpha(f \circ \gamma_2 \circ \phi \circ \alpha_1) && \text{by def. of } \alpha \\
&= \alpha_2 \circ f \circ \gamma_2 \circ \phi \circ \alpha_1 \circ \gamma_1 && \text{since } \alpha_2 \circ f \circ \gamma_2 \sqsubseteq f^\# \\
&\sqsubseteq f^\# \circ \phi \circ \alpha_1 \circ \gamma_1 && \text{since } \alpha_1 \circ \gamma_1 \text{ is reductive} \\
&\sqsubseteq f^\# \circ \phi && \text{by def. of } \star_{L_1^\#} f^\# \\
&= (\star_{L_1^\#} f^\#)(\phi)
\end{aligned}$$

Hence,  $\alpha \circ (\star_{L_1} f) \circ \gamma \sqsubseteq \star_{L_1^\#} f^\#$ .

Now consider (2). When  $\alpha_2 \circ f \circ \gamma_2 = f^\#$  and  $\alpha_1 \circ \gamma_1$  is the identity function,  $\sqsubseteq$  becomes  $=$  in the proof for (1).  $\square$

The following result states that performing the parametric analysis with a parametrized input and then binding the parameters to abstract properties yields the same result as the base analysis performed with the instantiation of the input with the same binding.

**Theorem 3.** Let  $f : D \xrightarrow{m} D$  and  $\kappa : L \xrightarrow{m} D$ . Then, for any  $\ell \in L$ ,

$$\text{lfp}_{\kappa(\ell)} f = (\text{lfp}_\kappa (\star_L f))(\ell)$$

*Proof.*  $(\text{lfp}_\kappa (\star_L f))(\ell) = (\bigsqcup_\beta (\star_L f)^{(\beta)}(\kappa))(\ell) = (\bigsqcup_\beta f^{(\beta)} \circ \kappa)(\ell) = \bigsqcup_\beta (f^{(\beta)} \circ \kappa)(\ell) = \bigsqcup_\beta (f^{(\beta)}(\kappa(\ell))) = \text{lfp}_{\kappa(\ell)} f$ .

*Remark 1.* In fact, any fixpoint of  $\star_L f$  provides a set of fixpoints of  $f$ . Let  $f : D \xrightarrow{m} D$  and  $\kappa : L \xrightarrow{m} D$  such that  $\kappa = (\star_L f)(\kappa)$ . Then, for any  $\ell \in L$ ,  $\text{lfp}_{\kappa(\ell)} f = \kappa(\ell)$  since  $f(\kappa(\ell)) = f \circ \kappa(\ell) = ((\star_L f)(\kappa))(\ell) = \kappa(\ell)$ .

## 4 Parametrizing Groundness Analysis

In logic programming, a value is a term that may contain variables. In any program state during the execution of a logic program, logic variables are bound to terms that may be in turn bound to other terms later during execution. A variable is ground in a substitution (program state) if the substitution maps the variable to a term that does not contain any variable. Groundness analysis is one of the most studied properties for logic programs [2,5,6,11,23,24]. This section presents a parametric groundness analysis by parametrizing the groundness analysis using positive propositional formulas with the simplest groundness domain.

### 4.1 Propositional Formulas

Let  $U$  be a finite set of propositional variables. A propositional formula over  $U$  is formed of propositional constants 0 and 1, propositional variables from  $U$

and logical connectives  $\wedge$ ,  $\vee$ ,  $\leftrightarrow$  and  $\neg$ . Other connectives such as  $\rightarrow$  and  $\leftarrow$  can be defined using these connectives. Let  $Bool = \{0, 1\}$  ordered by  $0 \leq 1$ . A truth substitution  $m$  on domain  $U$  is a partial function from  $U$  to  $Bool$ . The application of  $m$  to  $f$  is denoted  $m(f)$ . Let  $m = \{x \mapsto 1\}$  and  $f = (x \rightarrow y)$ . Then  $m(f) = (1 \rightarrow y)$ . If a truth substitution  $m$  is defined for every propositional variables in a propositional formula  $f$  then  $m$  is called a truth assignment for  $f$ . Given a formula  $f$  and a truth assignment  $m$ ,  $m \models f$  means that  $m$  satisfies  $f$  and  $f_1 \models f_2$  means that  $m \models f_1$  implies  $m \models f_2$  for every truth assignment  $m$  for  $f_1$ . Two formulas  $f_1$  and  $f_2$  are equivalent, denoted  $f_1 = f_2$  if both  $f_1 \models f_2$  and  $f_2 \models f_1$ . We shall not distinguish between elements in an equivalence class of  $=$ . A propositional formula  $f$  is positive if  $\mathbf{u} \models f$  for each such truth substitution  $\mathbf{u}$  that assigns 1 to all the propositional variables in  $f$ .

## 4.2 Groundness analysis

Marriott and Sondergaard [23] proposed to use positive propositional formulas to capture groundness dependencies between variables in a program state. Let  $x, y \in V$ . Then the formula  $x$  describes those program states in which  $x$  is bound to a ground term while  $x \rightarrow y$  describes those program states in which  $y$  is ground whenever  $x$  is. Let  $\mathcal{Pos}_V$  denotes the set of positive propositional formulas over propositional variables in  $V$ . Then  $\langle \mathcal{Pos}_V, \models \rangle$  is a complete lattice with bottom  $\wedge V$ , top 1, meet  $\wedge$  and join  $\vee$ . Let  $ground_V$  be defined  $ground_V(\theta) = \lambda x \in V. (\mathbf{V}(\theta(x)) = \emptyset)$  and

$$\begin{aligned} \alpha_{\mathcal{Pos}_V}(\theta) &= \bigvee_{\theta \in \Theta} \exists_{-V}. \bigwedge_{x \in \text{dom}(\theta)} (x \leftrightarrow \wedge \mathbf{V}(\theta(x))) \\ \gamma_{\mathcal{Pos}_V}(f) &= \{\theta \mid (ground_V(\theta) \models f)\} \end{aligned}$$

Then  $\langle \wp(\text{Subst}_V), \alpha_{\mathcal{Pos}_V}, \mathcal{Pos}_V, \gamma_{\mathcal{Pos}_V} \rangle$  is a Galois insertion [6]. Thus, the least upper bound  $\vee$  on  $\langle \mathcal{Pos}_V, \models \rangle$  approximates correctly  $\cup$  on  $\langle \wp(\text{Subst}_V), \subseteq \rangle$ . The other abstract operations for groundness analysis are given as follows. The abstract projection operation  $\pi_X^\sharp : \mathcal{Pos}_V \mapsto \mathcal{Pos}_{V \setminus X}$  is  $\pi_X^\sharp(f) = \exists x_1. \exists x_2. \dots \exists x_n. f$  when  $X = \{x_1, x_2, \dots, x_n\}$ ; the abstract unification operation  $\odot^\sharp : \mathcal{Pos}_U \times \mathcal{Pos}_V \mapsto \mathcal{Pos}_{U \cup V}$  is  $f_U \odot^\sharp f_V = f_U \wedge f_V$  and the abstract renaming operation  $\mathcal{R}_{\mathbf{x} \mapsto \mathbf{y}}^\sharp : \mathcal{Pos}_V \mapsto \mathcal{Pos}_{V \setminus \mathbf{x} \cup \mathbf{y}}$  is defined  $\mathcal{R}_{\mathbf{x} \mapsto \mathbf{y}}^\sharp(f) = f'$  where  $f'$  is obtained by simultaneously replacing the elements of  $\mathbf{x}$  with their corresponding elements in  $\mathbf{y}$ . For instance,  $\mathcal{R}_{x_1 x_2 \mapsto x_2 x_1}^\sharp(x_1 \rightarrow x_2) = (x_2 \rightarrow x_1)$ . The soundness of these operations are well established (see, e.g. [2]).

## 4.3 Abstract domain $\mathcal{G}_{\mathbb{P}}$

Jones and Sondergaard [17] proposed an abstract domain that capture groundness information in a substitution in terms of the collection of the variables that are grounded by the substitution. Let  $\mathbb{P}$  be the set of variables of interest.

The above abstract domain is isomorphic to the set of conjunctive propositional formulae with propositional variables from  $\mathbb{P}$

$$\mathcal{G}_{\mathbb{P}} = \{\wedge X \mid X \subseteq \mathbb{P}\}$$

ordered by logical implication  $\models$ . The partial order  $\langle \mathcal{G}_{\mathbb{P}}, \models \rangle$  is a complete lattice with bottom  $\wedge \mathbb{P}$ , top 1, meet  $\wedge$  and join  $\dot{\vee}$  where  $f_1 \dot{\vee} f_2 = \wedge \{f \mid f_1 \models f \text{ and } f_2 \models f\}$ . The abstraction and concretization functions are

$$\begin{aligned} \alpha_{\mathcal{G}_{\mathbb{P}}}(\Theta) &= \wedge \{x \mid x \in \mathbb{P} \text{ and } \forall \theta \in \Theta. (\mathbf{V}(\theta(x)) = \emptyset)\} \\ \gamma_{\mathcal{G}_{\mathbb{P}}}(\wedge X) &= \{\theta \mid \forall x \in X. (\mathbf{V}(\theta(x)) = \emptyset)\} \end{aligned}$$

$\langle \wp(\text{Subst}_{\mathbb{P}}), \alpha_{\mathcal{G}_{\mathbb{P}}}, \mathcal{G}_{\mathbb{P}}, \gamma_{\mathcal{G}_{\mathbb{P}}} \rangle$  is a Galois insertion.

#### 4.4 Parametrizing Groundness Analysis

A parametric analysis informs about how the abstract property at a program point depends on that at an initial program point. The parametric groundness analysis is obtained by parametrizing the abstract interpretation for groundness analysis with the groundness domain  $\mathcal{G}_{\mathbb{P}}$  where  $\mathbb{P}$  is the set of groundness parameters for the variables at the initial program point. The primitive abstract domains for the parametric analysis is thus  $\mathcal{G}_{\mathbb{P}} \xrightarrow{m} \mathcal{Pos}_U$  where  $U \in \mathcal{U}$ . The following abstract operations for the parametric analysis are lifted from those for the non-parametric groundness analysis. We shall drop the subscript in  $\star_{\mathcal{G}_{\mathbb{P}}}$ . By definition 1,

$$\begin{aligned} \phi_1(\star \wedge) \phi_2 &= \lambda g. (\phi_1(g) \wedge \phi_2(g)) \\ \phi_1(\star \vee) \phi_2 &= \lambda g. (\phi_1(g) \vee \phi_2(g)) \\ \star \pi_X^{\#}(\phi) &= \pi_X^{\#} \circ \phi \\ \star \mathcal{R}_{x \mapsto y}^{\#}(\phi) &= \mathcal{R}_{x \mapsto y}^{\#} \circ \phi \end{aligned}$$

### 5 Encoding Parametric Groundness Analysis

In this section, we encode monotone functions in  $\mathcal{G}_{\mathbb{P}} \xrightarrow{m} \mathcal{Pos}_U$  as positive propositional formulas in  $\mathcal{Pos}_{U \cup \mathbb{P}}$ . A monotone function  $\phi$  is encoded as a formula  $\nabla(\phi)$ . This encoding enables us to encode abstract operations on  $\mathcal{G}_{\mathbb{P}} \xrightarrow{m} \mathcal{Pos}_U$  in a straightforward manner. It turns out that the encoding of an abstract operation on  $\mathcal{G}_{\mathbb{P}} \xrightarrow{m} \mathcal{Pos}_U$  is exactly the corresponding operation on  $\mathcal{Pos}_{U \cup \mathbb{P}}$ .

*Encoding of abstract properties* Let  $g \in \mathcal{G}_{\mathbb{P}}$ . Then models of  $g$  are closed under conjunction, that is,  $M_1 \models g$  and  $M_2 \models g$  implies  $(M_1 \wedge M_2) \models g$  [6]. Thus,  $g$  has a minimum model which is the conjunction of all its models. The minimum model of  $g$  is denoted  $MM_{\mathbb{P}}(g)$ .

$$MM_{\mathbb{P}}(g) = \lambda x \in \mathbb{P}. \bigwedge \{m(x) \mid m \in (\mathbb{P} \mapsto \text{Bool}) \text{ and } (m \models g)\}$$



Let  $BF_{\mathbb{P}}(m)$  be the propositional formula over propositions in  $\mathbb{P}$  that has  $m$  as its minimal model. The formula is unique modulo logical equivalence.

$$BF_{\mathbb{P}}(m) = \left( \bigwedge_{u \in \mathbb{P}, m(u)=1} u \right) \wedge \left( \bigwedge_{u \in \mathbb{P}, m(u)=0} \neg u \right)$$

For instance  $BF_{\{u_1, u_2\}}(\{u_1 \mapsto 1, u_2 \mapsto 0\}) = u_1 \wedge \neg u_2$ .

*Example 1.* Let  $\mathbb{P} = \{\alpha\}$ . Then  $\mathcal{G}_{\mathbb{P}} = \{\alpha, 1\}$ .  $MM_{\mathbb{P}}(1) = \{\alpha \mapsto 0\}$  and  $MM_{\mathbb{P}}(\alpha) = \{\alpha \mapsto 1\}$ . Thus,  $BF_{\mathbb{P}}(MM_{\mathbb{P}}(1)) = \neg\alpha$  and  $BF_{\mathbb{P}}(MM_{\mathbb{P}}(\alpha)) = \alpha$ .

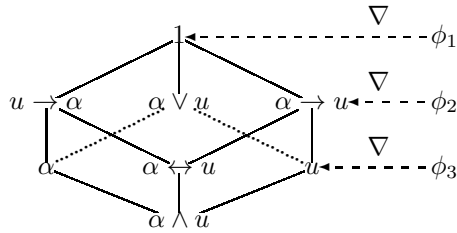
A function  $\phi$  from  $\mathcal{G}_{\mathbb{P}}$  to  $\mathcal{Pos}_U$  is represented as a formula in  $\mathcal{Pos}_{\mathbb{P} \cup U}$  via an embedding function  $\nabla$  defined as follows.

$$\nabla(\phi) = \bigvee_{g \in \mathcal{G}_{\mathbb{P}}} BF_{\mathbb{P}}(MM_{\mathbb{P}}(g)) \wedge \phi(g)$$

*Example 2.* Let  $\mathbb{P} = \{\alpha\}$  and  $U = \{u\}$ . Then  $\mathcal{G}_{\mathbb{P}} = \{\alpha, 1\}$  and  $\mathcal{Pos}_U = \{u, 1\}$ . There are four functions from  $\mathcal{G}_{\mathbb{P}}$  to  $\mathcal{Pos}_U$ :  $\phi_1 = \{\alpha \mapsto 1, 1 \mapsto 1\}$ ,  $\phi_2 = \{\alpha \mapsto u, 1 \mapsto 1\}$ ,  $\phi_3 = \{\alpha \mapsto u, 1 \mapsto u\}$  and  $\phi_4 = \{\alpha \mapsto 1, 1 \mapsto u\}$ . The first three functions are monotone and the last one is not. The embedding of the three monotone functions are as follows.

$$\begin{aligned} \nabla(\phi_1) &= 1 \\ \nabla(\phi_2) &= (\alpha \wedge u) \vee ((\neg\alpha) \wedge 1) = (\alpha \rightarrow u) \\ \nabla(\phi_3) &= u \end{aligned}$$

Applying  $\nabla$  to  $\phi_4$ , we obtain  $\nabla(\phi_4) = \alpha \vee u$ . The following diagram shows  $\mathcal{Pos}_{\mathbb{P} \cup U}$  and encoding of monotone functions  $\mathcal{G}_{\mathbb{P}} \xrightarrow{m} \mathcal{Pos}_U$  via  $\nabla$ .



There are positive propositional formulas in  $\mathcal{Pos}_{\mathbb{P} \cup U}$  such as  $\nabla(\phi_4)$  that are not images of monotone functions in  $\mathcal{G}_{\mathbb{P}} \xrightarrow{m} \mathcal{Pos}_U$  under  $\nabla$ . These formulas are not used in the parametric analysis.

**Lemma 1.**  $\nabla$  is monotone and 1-1.

*Proof.* That  $\nabla$  is monotone follows from its definition straightforwardly. We now prove that  $\nabla$  is 1-1. Let  $\phi_1 \neq \phi_2$ . Then there is  $g$  such that  $\phi_1(g) \neq \phi_2(g)$

implying there is a truth assignment  $m : U \mapsto \text{Bool}$  such that  $m(\phi_1(g)) \neq m(\phi_2(g))$ . Construct a truth assignment  $m' : \mathbb{P} \cup U \mapsto \text{Bool}$  as follows.

$$m'(y) = \begin{cases} MM_{\mathbb{P}}(g)(y) & \text{if } y \in \mathbb{P} \\ m(y) & \text{otherwise} \end{cases}$$

Then  $m'(\nabla(\phi_1)) = m'(\phi_1(g)) = m(\phi_1(g))$  since  $\phi_1(g)$  does not contain any propositional variable in  $\mathbb{P}$ . Similarly,  $m'(\nabla(\phi_2)) = m'(\phi_2(g)) = m(\phi_2(g))$ . Thus,  $\nabla(\phi_1) \neq \nabla(\phi_2)$ .  $\square$

*Decoding abstract properties and instantiating analysis* Since  $\nabla$  is 1-1, its inverse exists. Define  $\nabla^{-1}(h) = \lambda g. MM_{\mathbb{P}}(g)(h)$ . The following lemma proves that  $\nabla^{-1}$  is the inverse of  $\nabla$ .

**Lemma 2.**  $\nabla^{-1}(\nabla(\phi)) = \phi$  for any function in  $\mathcal{G}_{\mathbb{P}} \xrightarrow{m} \mathcal{Pos}_U$ .

*Proof.* Note that  $MM_{\mathbb{P}}(g)(BF_{\mathbb{P}}(MM_{\mathbb{P}}(g')))) = 0$  for any  $g' \neq g$ . Hence,  $(\nabla^{-1} \circ \nabla(\phi))(g) = MM_{\mathbb{P}}(g)(\nabla(\phi)) = \phi(g)$ .  $\square$

Instantiating an analysis result  $\nabla(\phi)$  for a given input  $g$  amounts to calculating  $\phi(g)$  which, according to the above proof, amounts to calculating  $MM_{\mathbb{P}}(g)(\nabla(\phi))$ . Thus, instantiating an analysis result for a given input  $g$  does not requires a complete decoding.

*Encoding Analysis Input* Let  $V$  be the set of variables in the initial goal. The parametric analysis can be performed with any monotone function  $\mathcal{G}_{\mathbb{P}} \xrightarrow{m} \mathcal{Pos}_V$  as input. A more natural input associates each variable in  $V$  with a different parameter since it allows the influence of the groundness of the variables in the initial goal on groundness dependencies at other program points to be observed. The following shows that the input has a natural encoding. Define  $BM_{\mathbb{P}}(X) = BF_{\mathbb{P}}(MM_{\mathbb{P}}(\bigwedge X))$  for any  $X \subseteq \mathbb{P}$ .

**Theorem 4.** Let  $|V| = |\mathbb{P}|$ ,  $\rho : \mathbb{P} \mapsto V$  an invertible function and  $\iota : \mathcal{G}_{\mathbb{P}} \xrightarrow{m} \mathcal{Pos}_V$  defined  $\iota(\bigwedge X) = \bigwedge \{\rho(x) \mid x \in X\}$ . Then  $\nabla(\iota) = \bigwedge_{\alpha \in \mathbb{P}} (\alpha \rightarrow \rho(\alpha))$ .

*Proof.* The proof is by induction on  $|\mathbb{P}|$ .

*Basis .* The thesis holds vacuously for the case  $|\mathbb{P}| = 0$ .

*Induction .* Assume that thesis holds for all  $\mathbb{P}$  such that  $|\mathbb{P}| = n$ . Assume that  $|\mathbb{P}'| = n + 1$ . There are  $\alpha$  and  $\mathbb{P}$  such that  $\mathbb{P}' = \mathbb{P} \cup \{\alpha\}$  and  $\alpha \notin \mathbb{P}$ .

$$\begin{aligned} \nabla(\iota) &= \bigvee_{g \in \mathcal{G}_{\mathbb{P}'}} BF_{\mathbb{P}'}(MM_{\mathbb{P}'}(g)) \wedge \iota(g) \\ &= \bigvee_{X \in \wp(\mathbb{P}')} BM_{\mathbb{P}'}(X) \wedge \iota(\bigwedge X) \\ &= \bigvee_{X \in \wp(\mathbb{P})} BM_{\mathbb{P}'}(X) \wedge \iota(\bigwedge X) \vee \bigvee_{X \in \wp(\mathbb{P})} BM_{\mathbb{P}'}(X \cup \{\alpha\}) \wedge \iota(\alpha \wedge \bigwedge X) \end{aligned}$$

$$\begin{aligned}
&= (\neg\alpha) \wedge \bigvee_{X \in \wp(\mathbb{P})} BM_{\mathbb{P}}(X) \wedge \iota(\bigwedge X) \vee \alpha \wedge \rho(\alpha) \bigvee_{X \in \wp(\mathbb{P})} BM_{\mathbb{P}}(X) \wedge \iota(\bigwedge X) \\
&= (\alpha \rightarrow \rho(\alpha)) \wedge \bigvee_{X \in \wp(\mathbb{P})} BM_{\mathbb{P}}(X) \wedge \iota(\bigwedge X) \\
&= (\alpha \rightarrow \rho(\alpha)) \wedge \bigwedge_{\beta \in \mathbb{P}} (\beta \rightarrow \rho(\beta)) \text{ by the induction hypothesis} \\
&= \bigwedge_{\beta \in \mathbb{P}'} (\beta \rightarrow \rho(\beta))
\end{aligned}$$

Hence the thesis holds for  $\mathbb{P}'$ .  $\square$

*Example 3.* Let  $U = \{x_1, x_2\}$  and  $\mathbb{P} = \{\alpha_1, \alpha_2\}$ . The encoding of the monotone function  $\{\alpha_1 \mapsto x_1, \alpha_2 \mapsto x_2, \alpha_1\alpha_2 \mapsto x_1 \wedge x_2, 1 \mapsto 1\}$  is  $(\alpha_1 \rightarrow x_1) \wedge (\alpha_2 \rightarrow x_2)$ .

*Encoding abstract operations* The encoding  $\nabla$  allows us to use the same set of the operations for both non-parametric and parametric groundness analyses, which is formally stated in the following theorem. The theorem also states that  $\nabla(\mathcal{G}_{\mathbb{P}} \xrightarrow{m} \mathcal{Pos}_X)$  is closed under all the analysis operations.

**Theorem 5.** Let  $\mathbb{P}$  be a set of parameters,  $U, V \in \mathcal{U}$ ,  $\mathbf{x}$  and  $\mathbf{y}$  be vectors of variables such that  $|\mathbf{x}| = |\mathbf{y}|$ . Then

1.  $\nabla(\phi_1(\star\wedge)\phi_2) = \nabla(\phi_1) \wedge \nabla(\phi_2)$  for any  $\phi_1 \in (\mathcal{G}_{\mathbb{P}} \xrightarrow{m} \mathcal{Pos}_U)$  and  $\phi_2 \in (\mathcal{G}_{\mathbb{P}} \xrightarrow{m} \mathcal{Pos}_V)$ ;
2.  $\nabla(\phi_1(\star\vee)\phi_2) = \nabla(\phi_1) \vee \nabla(\phi_2)$  for any  $\phi_1, \phi_2 \in (\mathcal{G}_{\mathbb{P}} \xrightarrow{m} \mathcal{Pos}_U)$ ;
3.  $\nabla(\star\pi_X^{\sharp}(\phi)) = \pi_X^{\sharp}(\nabla(\phi))$  for any  $\phi \in (\mathcal{G}_{\mathbb{P}} \xrightarrow{m} \mathcal{Pos}_U)$  and any  $X \subseteq U$ ;
4.  $\nabla(\star\mathcal{R}_{\mathbf{x} \mapsto \mathbf{y}}^{\sharp}(\phi)) = \mathcal{R}_{\mathbf{x} \mapsto \mathbf{y}}^{\sharp}(\nabla(\phi))$  for any  $\phi \in (\mathcal{G}_{\mathbb{P}} \xrightarrow{m} \mathcal{Pos}_U)$ .

*Proof.* Consider (1) first. Note that  $BM_{\mathbb{P}}(X) \wedge BM_{\mathbb{P}}(Y) = 0$  when  $X \neq Y$ .

$$\begin{aligned}
\nabla(\phi_1) \wedge \nabla(\phi_2) &= (\bigvee_{X \in \wp(\mathbb{P})} BM_{\mathbb{P}}(X) \wedge \phi_1(\bigwedge X)) \wedge (\bigvee_{Y \in \wp(\mathbb{P})} BM_{\mathbb{P}}(Y) \wedge \phi_2(\bigwedge Y)) \\
&= \bigvee_{X \in \wp(\mathbb{P}), Y \in \wp(\mathbb{P})} BM_{\mathbb{P}}(X) \wedge BM_{\mathbb{P}}(Y) \wedge \phi_1(\bigwedge X) \wedge \phi_2(\bigwedge Y) \\
&= \bigvee_{X \in \wp(\mathbb{P})} BM_{\mathbb{P}}(X) \wedge \phi_1(\bigwedge X) \wedge \phi_2(\bigwedge X) \\
&= \nabla(\lambda g. \phi_1(g) \wedge \phi_2(g)) \\
&= \nabla(\phi_1(\star\wedge)\phi_2)
\end{aligned}$$

The proof of (2) is similar. (3) and (4) are straightforward.  $\square$

The following theorem shows that encoding of a monotone function  $\phi$  is logically equivalent to  $\bigwedge_{g \in \mathcal{G}_{\mathbb{P}}} (g \rightarrow \phi(g))$ .

**Theorem 6.** For any  $\mathbb{P}, V$  such that  $\mathbb{P} \cap V = \emptyset$ ,

$$\forall \phi \in \mathcal{G}_{\mathbb{P}} \xrightarrow{m} \mathcal{Pos}_V. \left( \bigvee_{X \in \wp(\mathbb{P})} BM_{\mathbb{P}}(X) \wedge \phi(\bigwedge X) = \bigwedge_{X \in \wp(\mathbb{P})} (\bigwedge X \rightarrow \phi(\bigwedge X)) \right) \quad (1)$$

*Proof.* The proof is done by induction on cardinality of  $\mathbb{P}$ .

*Basis.*  $|\mathbb{P}| = 0$  and hence  $\mathbb{P} = \emptyset$ . Then  $\wp(\mathbb{P}) = \{\emptyset\}$ . Thus,  $\bigvee_{X \in \wp(\mathbb{P})} BM_{\mathbb{P}}(X) \wedge \phi(\bigwedge X) = BM_{\emptyset}(\emptyset) \wedge \phi(\bigwedge \emptyset) = \phi(1)$ . We also have  $\bigwedge_{X \in \wp(\mathbb{P})} (\bigwedge X \rightarrow \phi(\bigwedge X)) = \bigwedge \emptyset \rightarrow \phi(\bigwedge \emptyset) = \phi(1)$ . Hence, formula 1 holds for the base case.

*Induction.* Assume that formula 1 holds for any  $\mathbb{P}$  such that  $|\mathbb{P}| = n$  and  $\mathbb{P} \cap V = \emptyset$ . Let  $\mathbb{P}' = \mathbb{P} \cup \{z\}$  where  $z \notin \mathbb{P}$  is an arbitrary variable and  $\phi$  be an arbitrary monotone in  $\mathcal{G}_{\mathbb{P}'}$   $\xrightarrow{m}$   $\mathcal{Pos}_V$ . Then  $|\mathbb{P}'| = n + 1$ . Note that  $\wp(\mathbb{P}') = \wp(\mathbb{P}) \cup \{Y \cup \{z\} \mid Y \in \wp(\mathbb{P})\}$ . Then

$$\begin{aligned} & \bigwedge_{X \in \wp(\mathbb{P}')} (\bigwedge X \rightarrow \phi(\bigwedge X)) \\ &= \left( \bigwedge_{X \in \wp(\mathbb{P})} (\bigwedge X \rightarrow \phi(\bigwedge X)) \right) \wedge \left( \bigwedge_{Y \in \wp(\mathbb{P})} (z \wedge \bigwedge Y \rightarrow \phi(z \wedge \bigwedge Y)) \right) \\ &= \left( \bigwedge_{X \in \wp(\mathbb{P})} (\bigwedge X \rightarrow \phi(\bigwedge X)) \right) \wedge \left( \neg z \vee \bigwedge_{Y \in \wp(\mathbb{P})} (\bigwedge Y \rightarrow \phi(z \wedge \bigwedge Y)) \right) \\ &= \left( \bigwedge_{X \in \wp(\mathbb{P})} (\bigwedge X \rightarrow \phi(\bigwedge X)) \right) \wedge \left( \neg z \vee \bigwedge_{Y \in \wp(\mathbb{P})} (\bigwedge Y \rightarrow \phi'(\bigwedge Y)) \right) \end{aligned}$$

where  $\phi'(\bigwedge Y) = \phi(z \wedge \bigwedge Y)$  for all  $Y \in \wp(\mathbb{P})$ . Since  $\phi \in \mathcal{G}_{\mathbb{P}'} \xrightarrow{m} \mathcal{Pos}_V$  and  $z \notin \mathbb{P}$ , both  $\phi \in \mathcal{G}_{\mathbb{P}} \xrightarrow{m} \mathcal{Pos}_V$  and  $\phi' \in \mathcal{G}_{\mathbb{P}} \xrightarrow{m} \mathcal{Pos}_V$ . By applying the induction hypothesis twice, we have

$$\begin{aligned} & \bigwedge_{X \in \wp(\mathbb{P}')} (\bigwedge X \rightarrow \phi(\bigwedge X)) \\ &= \left( \bigvee_{X \in \wp(\mathbb{P})} BM_{\mathbb{P}}(X) \wedge \phi(\bigwedge X) \right) \wedge \left( \neg z \vee \bigvee_{Y \in \wp(\mathbb{P})} (BM_{\mathbb{P}}(Y) \wedge \phi'(\bigwedge Y)) \right) \\ &= \left( \bigvee_{X \in \wp(\mathbb{P})} (\neg z \wedge BM_{\mathbb{P}}(X) \wedge \phi(\bigwedge X)) \right. \\ & \quad \left. \bigvee \bigvee_{X \in \wp(\mathbb{P}), Y \in \wp(\mathbb{P})} (BM_{\mathbb{P}}(X) \wedge BM_{\mathbb{P}}(Y) \wedge \phi(\bigwedge X) \wedge \phi'(\bigwedge Y)) \right) \end{aligned}$$

Since  $z \notin \mathbb{P}$ ,  $\neg z \wedge BM_{\mathbb{P}}(X) = BM_{\mathbb{P} \cup \{z\}}(X) = BM_{\mathbb{P}'}(X)$  for any  $X \in \wp(\mathbb{P})$ . Suppose  $X, Y \in \wp(\mathbb{P})$  and  $X \neq Y$ . Then there is a  $v \in \mathbb{P}$  such that (i)  $v \in X \setminus Y$  or (ii)  $v \in Y \setminus X$ . Consider the case (i) and let  $X_v = X \setminus \{v\}$  and

$\mathbb{P}_v = \mathbb{P} \setminus \{v\}$ .  $BM_{\mathbb{P}}(X) \wedge BM_{\mathbb{P}}(Y) = \neg(\bigvee(\mathbb{P} \setminus X)) \wedge \bigwedge X \wedge \neg(\bigvee(\mathbb{P} \setminus Y)) \wedge \bigwedge Y = \neg(\bigvee(\mathbb{P} \setminus X)) \wedge \bigwedge X_v \wedge v \wedge \neg v \neg(\bigvee(\mathbb{P}_v \setminus Y)) \wedge \bigwedge Y = 0$ . *Similarly,  $BM_{\mathbb{P}}(X) \wedge BM_{\mathbb{P}}(Y) = 0$  in the case (ii).* By monotonicity of  $\phi$ ,  $\phi(\bigwedge X) \wedge \phi'(\bigwedge X) = \phi(\bigwedge X) \wedge \phi(z \wedge \bigwedge X) = \phi(z \wedge \bigwedge X)$  and  $\phi(\bigwedge X) \vee \phi(z \wedge \bigwedge X) = \phi(\bigwedge X)$  for any  $X \in \wp(\mathbb{P})$ . Then,

$$\begin{aligned}
& \bigwedge_{X \in \wp(\mathbb{P}')} (\bigwedge X \rightarrow \phi(\bigwedge X)) \\
&= \bigvee_{X \in \wp(\mathbb{P})} (BM_{\mathbb{P}'}(X) \wedge \phi(\bigwedge X)) \vee \bigvee_{X \in \wp(\mathbb{P})} (BM_{\mathbb{P}}(X) \wedge \phi(\bigwedge X) \wedge \phi'(\bigwedge X)) \\
&= \bigvee_{X \in \wp(\mathbb{P})} (BM_{\mathbb{P}'}(X) \wedge \phi(\bigwedge X)) \vee \bigvee_{X \in \wp(\mathbb{P})} (BM_{\mathbb{P}}(X) \wedge \phi(z \wedge \bigwedge X)) \\
&= \bigvee_{X \in \wp(\mathbb{P})} (BM_{\mathbb{P}'}(X) \wedge \phi(\bigwedge X)) \vee \bigvee_{X \in \wp(\mathbb{P})} (\neg z \wedge BM_{\mathbb{P}}(X) \wedge \phi(z \wedge \bigwedge X)) \\
&\quad \vee \bigvee_{X \in \wp(\mathbb{P})} (z \wedge BM_{\mathbb{P}}(X) \wedge \phi(z \wedge \bigwedge X)) \\
&= \bigvee_{X \in \wp(\mathbb{P})} (BM_{\mathbb{P}'}(X) \wedge \phi(\bigwedge X)) \vee \bigvee_{X \in \wp(\mathbb{P})} (BM_{\mathbb{P}'}(X) \wedge \phi(z \wedge \bigwedge X)) \\
&\quad \vee \bigvee_{X \in \wp(\mathbb{P})} (BM_{\mathbb{P}'}(X \cup \{z\}) \wedge \phi(\bigwedge(X \cup \{z\}))) \\
&= \bigvee_{X \in \wp(\mathbb{P})} (BM_{\mathbb{P}'}(X) \wedge (\phi(\bigwedge X) \vee \phi(z \wedge \bigwedge X))) \\
&\quad \vee \bigvee_{X \in \wp(\mathbb{P})} (BM_{\mathbb{P}'}(X \cup \{z\}) \wedge \phi(\bigwedge(X \cup \{z\}))) \\
&= \bigvee_{X \in \wp(\mathbb{P})} (BM_{\mathbb{P}'}(X) \wedge \phi(\bigwedge X)) \vee \bigvee_{X \in \wp(\mathbb{P})} (BM_{\mathbb{P}'}(X \cup \{z\}) \wedge \phi(\bigwedge(X \cup \{z\}))) \\
&= \bigvee_{X \in \wp(\mathbb{P}')} (BM_{\mathbb{P}'}(X) \wedge \phi(\bigwedge X))
\end{aligned}$$

□

## 6 Prototype Implementation

We have implemented a logic program analyzer in SICSTus Prolog and the CUDD package that can perform both parametric and non-parametric groundness analysis. The analyzer takes a text file as input that contains a Prolog program, a directive of the form `:- main(Pred/Arity)` specifying a top-level goal and a directive `:- parametric(yes)` if the parametric analysis is to be performed.

## 6.1 Analysis Process

The analyzer first does the magic transformation [12] on the source program and the top-level goal  $q(x_1, \dots, x_n)$  that is constructed from the directive `:-main(q/n)`. For each predicate  $p/n$  in the source program, the transformed program contains two predicates  $call\_p/n$  and  $ans\_p/n$  such that success sets for  $call\_p/n$  and  $ans\_p/n$  are the set of calls to  $p$  and the set of successes of  $p$  in the source program during the execution of the top-level goal. In the second step, the analyzer constructs a call-graph which captures dependencies between the clauses of the transformed program and computes strongly connected components (SCCs) of the call-graph. The third step normalizes the transformed program and then abstractly compiles [15] the normalized program by replacing each built-in with its success pattern. For instance,  $x_1 = x_3$  is replaced with  $x_1 \leftrightarrow x_3$ . Then, the unit clause  $call\_a(x_1, \dots, x_n)$  is added for the non-parametric analysis or the clause  $call\_a(x_1, \dots, x_n) :- (\beta_1 \rightarrow x_1) \wedge \dots \wedge (\beta_n \rightarrow x_n)$  is added otherwise. Then the success pattern of the abstract program is computed according to the SCCs which yields call and success patterns for the source program and the top level goal. Note that SCCs are computed before abstract compilation. This is because abstract compilation incurs loss of concrete information, which may result in more dependencies between clauses.

Consider the reverse program with top-level goal  $r(x_1, x_2)$ . Suppose that we want to perform the parametric analysis. Then the text file contains.

$$:- main(r/2). \quad (2)$$

$$:- parametric(yes). \quad (3)$$

$$r([], []). \quad (4)$$

$$r([x_1|x_2], x_3) :- r(x_2, x_4), a(x_4, [x_1], x_3). \quad (5)$$

$$a([], x, x). \quad (6)$$

$$a([x_1|x_2], x_3, [x_1|x_4]) :- a(x_2, x_3, x_4). \quad (7)$$

The following is the abstract program that is obtained where  $xy$  abbreviates  $x \wedge y$ .

$$call\_r(x_1, x_2) :- (\beta_1 \rightarrow x_1) \wedge (\beta_2 \rightarrow x_2). \quad (8)$$

$$ans\_r(x_1, x_2) :- call\_r(x_1, x_2), x_1x_2. \quad (9)$$

$$call\_r(x_4, x_5) :- call\_r(x_1, x_2), (x_1 \leftrightarrow x_3x_4). \quad (10)$$

$$call\_a(x_5, x_6, x_2) :- call\_r(x_1, x_2), (x_1 \leftrightarrow x_3x_4), ans\_r(x_4, x_5), (x_6 \leftrightarrow x_3). \quad (11)$$

$$ans\_r(x_1, x_2) :- call\_r(x_1, x_2), (x_1 \leftrightarrow x_3x_4), ans\_r(x_4, x_5), \\ (x_6 \leftrightarrow x_3), ans\_a(x_5, x_6, x_2). \quad (12)$$

$$ans\_a(x_1, x_2, x_3) :- call\_a(x_1, x_2, x_3), x_1 \wedge (x_2 \leftrightarrow x_3). \quad (13)$$

$$call\_a(x_5, x_2, x_6) :- call\_a(x_1, x_2, x_3), (x_1 \leftrightarrow x_4x_5) \wedge (x_3 \leftrightarrow x_4x_6) \quad (14)$$

$$ans\_a(x_1, x_2, x_3) :- call\_a(x_1, x_2, x_3), (x_1 \leftrightarrow x_4x_5) \wedge (x_3 \leftrightarrow x_4x_6), \\ ans\_a(x_5, x_2, x_6). \quad (15)$$

Each clause in the abstract program is derived from the input file. The clause 8 results from the clauses 2 and 3, the clause 9 from the clause 4, the clauses 10,11 and 12 from the clause 5, the clauses 13 from the clause 6 and the clauses 14 and 15 from the clause 7. The SCCs are  $\{8\}$ ,  $\{9\}$ ,  $\{10\}$  and  $\{11, 12, 13, 14, 15\}$  with the latter SCCs depending only on the earlier ones. After evaluating the abstract program, we obtain

$$\begin{aligned} call\_a(x_1, x_2, x_3) &:- (\beta_1 \rightarrow x_1 x_2) \\ ans\_a(x_1, x_2, x_3) &:- (\beta_1 \rightarrow x_1 x_2) \wedge (x_3 \leftrightarrow x_1 x_2) \\ call\_r(x_1, x_2) &:- (\beta_1 \rightarrow x_1) \\ ans\_r(x_1, x_2) &:- (x_1 \leftrightarrow x_2) \wedge ((\beta_1 \vee \beta_2) \rightarrow x_1 x_2) \end{aligned}$$

The call pattern for  $r/2$  states that  $r/2$  is (recursively) called with the first argument being a ground term if the first argument of the top-level goal is ground ( $\beta_1 = 1$ ). There is no similar relationship between the second argument of a recursive call to  $r/2$  with the second argument of the top-level goal. This is precise since  $r/2$  is recursively called with its second argument being a fresh variable in the second clause for  $r/2$ . The success pattern for  $r(x_1, x_2)$  has two parts. The first part  $x_1 \leftrightarrow x_2$  is what a goal independent analysis infers and it states that upon success,  $x_1$  is ground iff  $x_2$  is. The second part captures the effect of the groundness parameters on the groundness of the arguments of the calls. It states that both  $x_1$  and  $x_2$  are ground if either argument of the top level goal is ground.

## 6.2 An Example

The following is the quicksort program plus analysis directives. The first directive indicates the top-level goal  $qs(x_1, x_2)$  and the second the parametric analysis. Thus, the input abstract property is  $(\beta_1 \rightarrow x_1) \wedge (\beta_2 \rightarrow x_2)$ .

```
:- main(qs/2).
:- parametric(yes).

app([], L, L).
app([X|L1], L2, [X|L3]) :- app(L1, L2, L3).

pt([X|T], P, [X|B], A) :- leq(X, P), pt(T, P, B, A).
pt([X|T], P, B, [X|A]) :- gt(X, P), pt(T, P, B, A).
pt([], _, [], []).

leq(X, Y) :- X <= Y.
gt(X, Y) :- X > Y.

qs([], []).
qs([X|Xs], Ys) :- pt(Xs, X, U, V), qs(U, S), qs(V, L), app(S, [X|L], Ys).
```

The predicates  $leq/2$  and  $gt/2$  have been added to observe the effect of groundness parameters on their arguments. The following is the analysis result

that has been converted manually to more readable form.

$$call\_gt(x_1, x_2) :- \beta_1 \rightarrow x_1 x_2 \quad (16)$$

$$ans\_gt(x_1, x_2) :- x_1 x_2 \quad (17)$$

$$call\_leq(x_1, x_2) :- \beta_1 \rightarrow x_1 x_2 \quad (18)$$

$$ans\_leq(x_1, x_2) :- x_1 x_2 \quad (19)$$

$$call\_pt(x_1, x_2, x_3, x_4) :- \beta_1 \rightarrow x_1 x_2 \quad (20)$$

$$ans\_pt(x_1, x_2, x_3, x_4) :- (\beta_1 \rightarrow x_2) \wedge x_1 x_3 x_4 \quad (21)$$

$$call\_qs(x_1, x_2) :- (\beta_1 \rightarrow x_1) \wedge (\beta_2 \rightarrow (x_1 \vee x_2)) \quad (22)$$

$$ans\_qs(x_1, x_2) :- (x_1 \leftrightarrow x_2) \wedge ((\beta_1 \vee \beta_2) \rightarrow x_1 x_2) \quad (23)$$

$$call\_app(x_1, x_2, x_3) :- x_1 \wedge (\beta_1 \rightarrow x_2) \wedge (\beta_2 \rightarrow (x_2 \vee x_3)) \quad (24)$$

$$ans\_app(x_1, x_2, x_3) :- x_1 \wedge (x_2 \leftrightarrow x_3) \wedge ((\beta_1 \vee \beta_2) \rightarrow x_2 x_3) \quad (25)$$

The analysis result gives call and success patterns during the execution of the top-level goal  $qs(x_1, x_2)$  using  $\beta_1$  for the groundness of  $x_1$  at the beginning of the execution and  $\beta_2$  for that of  $x_2$ . By assigning 1 to  $\beta_1$  in the righthand side of Eq. 16, we obtain  $x_1 x_2$ , implying that  $gt/2$  (hence  $>/2$ ) is always called with ground arguments if the first argument of the top-level goal is ground. Eq 17 indicates  $gt/2$  (and  $>/2$ ) always instantiates its arguments to ground terms. Call and success patterns for  $leq/2$  are the same as those for  $gt/2$ . This illustrates that the parametric analysis allows us to infer a sufficient groundness condition on the top-level goal for the execution of the program to avoid instantiation errors [18]. Eq. 20 indicates that if the first argument of the top-level goal is ground ( $\beta_1 = 1$ ) then  $pt(x_1, x_2, x_3, x_4)$  is always called with both  $x_1$  and  $x_2$  being ground. Eq 21 says that upon success,  $pt(x_1, x_2, x_3, x_4)$  binds  $x_1, x_3$  and  $x_4$  to ground terms and it binds  $x_2$  to a ground term if  $\beta_1 = 1$ . Observe that  $x_2$  may be any term when  $x_1, x_3$  and  $x_4$  are all empty lists.

The call pattern in Eq. 22 says that  $qs(x_1, x_2)$  is called with  $x_1$  ground if  $\beta_1 = 1$  and that either  $x_1$  or  $x_2$  is ground if  $\beta_2 = 1$ . The success pattern for  $qs(x_1, x_2)$  in Eq. 23 states that  $x_1$  is ground iff  $x_2$  is ground and that both  $x_1$  and  $x_2$  are ground if either  $\beta_1$  or  $\beta_2$  is ground. From Eq. 24, we can infer that when  $app(x_1, x_2, x_3)$  is called,  $x_1$  is always ground, and  $x_2$  is ground if  $\beta_1 = 1$ , and at least one of  $x_2$  and  $x_3$  is ground if  $\beta_2 = 1$ . From Eq. 25, one can deduce that upon success of  $app(x_1, x_2, x_3)$ ,  $x_1$  is always ground,  $x_2$  is ground iff  $x_3$  is ground, and both  $x_2$  and  $x_3$  are ground if either  $\beta_1$  or  $\beta_2$  is 1.

### 6.3 Performance

The analyzer has been tested with a suite of benchmark programs. The experiments were done on a 2.33GHz Intel (R) Xeon (R) CPU running Linux 2.6.24 and SICSTUS Prolog 4.0.3. The CUDD package version is 2.4.1.

Table 1 shows data from the experiment. All but the last row corresponds to a benchmark program. The first column contains the name of the program and the second specifies the top level goal. In the third column is the number



Program	Top-Level	Size	Para	Non-Para	Ratio
ann1	go/1	1570	273.68	271.57	1.00
asm	asm_PIL/2	3589	757.89	754.73	1.00
boyer	tautology/1	725	63.68	65.78	0.96
cs_r	pgenconfig/1	1101	146.31	140.52	1.04
disj_r	top/1	682	60.52	57.36	1.05
dnf	dnf/2	358	29.47	33.15	0.88
ga	test_ga/2	1349	176.31	166.84	1.05
gabriel	main/2	377	23.15	23.15	1.00
kalah	play/2	855	74.73	76.31	0.97
life	life/4	272	15.26	13.68	1.11
meta	interpret/1	201	14.73	11.05	1.33
nandc	play/1	486	32.10	31.05	1.03
nbody	go/2	1431	125.78	120.00	1.04
neural	test/2	755	69.47	70.00	0.99
peep	comppeepopt/3	1435	180.52	176.84	1.02
press	test_press/2	1303	241.57	232.10	1.04
read	read/2	1686	281.05	272.63	1.03
reducer	try/2	1063	137.36	123.68	1.11
ronp	puzzle/1	340	19.47	18.42	1.05
sdda	do_sdda/4	788	82.10	84.73	0.96
semi	go/2	1351	150.00	149.47	1.00
simple_analyzer	main/1	1537	242.63	238.42	1.01
tictactoe	play/1	474	34.73	32.10	1.08
tsp	tsp/5	391	30.52	25.78	1.18
zebra	zebra/7	259	18.42	10.52	1.75
Total		24378	3281.57	3199.99	1.02

**Table 1.** Performance Comparison between Parametric and Non-Parametric Analyses

of atoms in the abstract program. The fourth column is the time in millisecond spent on the parametric analysis using  $(\beta_1 \rightarrow x_1) \wedge \dots \wedge (\beta_n \rightarrow x_n)$  as the input abstract property. The fifth column contains the time spent on the non-parametric analysis which is performed without any input groundness information. The last column contains the ratio of the fourth over the fifth. The last row gives the total size, total times and the average ratio.

The table indicates that the prototype parametric groundness analyzer spends an average of 0.135 seconds to process one thousand atoms in the abstract program. This is an acceptable speed for most logic programs. The table shows that the time the parametric analysis takes is from 0.88 to 1.75 times that the non-parametric analysis takes with an average of 1.02. This indicates that extra cost is negligible for performing the parametric analysis which yields more general results, which is quite surprising and promising.

## 7 Related Work

The approach proposed in section 3 for parametrizing a base analysis lifts each primitive abstract domain of the base analysis to its cardinal power with an exponent over which parameters range. The cardinal power belongs to the standard Cousot and Cousot’s abstract interpretation theory and was proposed in [8] to capture dependencies between abstract properties of a concrete entity [14]. Let  $\langle D, \alpha_E, E^\sharp, \gamma_E \rangle$  and  $\langle D, \alpha_B, B^\sharp, \gamma_B \rangle$  be a Galois connection. Then  $\langle D, \alpha, E^\sharp \xrightarrow{m} B^\sharp, \gamma \rangle$  is a Galois connection where  $\alpha = \lambda d. \lambda e. (\alpha_B(d \sqcap_D \gamma_E(e)))$  and  $\gamma$  is that induced by  $\alpha$ . The cardinal power domain in [8] and the relative reduced power domain in [13] are refinements of the base domain. In contrast, we use cardinal power to capture dependency of analysis output on analysis input.

Parametric analysis abounds in literature. The following are a few examples. Chatterjee et. al. present a point-to analysis for typed object oriented languages [4]. This analysis computes a summary function for each method that expresses the effect of the method on the points-to solution. The summary function is parametrized by symbolic unknown initial values and conditions on these values. The actual-formal bindings are accounted for when points-to information is propagated into a method from its callers. Liang and Harrod uses symbolic names for memory locations whose addresses may be passed into a procedure [21]. These symbolic names are then used in point-to graphs which expresses parametrized summary information for a procedure. The summary information can then be instantiated at specific call sites by binding the symbolic names. The escape analysis by Blanchet [3] is a combination of forward and backward analysis. The backward analysis computes escape information for method arguments as a function of the escape information for method result. These bespoke analyses were not designed by parametrizing a base analysis. Abstract properties in these analyses are functions over parameters; thus it is interesting to study whether and how they can be designed by parametrizing a base analysis.

In [22] is a groundness analysis of logic programs that is also parametrized by a number of groundness parameters. The analysis is designed from Jones and Sondergaard’s analysis by lazily evaluating operations on groundness parameters. However, it does not capture groundness dependencies precisely between variables in the program compared with the parametric groundness analysis presented in this paper. Moreover, the extra cost of performing that analysis over the corresponding non-parametric analysis is 78% which is significant.

This paper shows by an example that inference of sufficient groundness condition for error free execution can be done with a traditional top down forward analysis framework. One benefit that comes with a top down analysis is that analysis can be made more precise because of availability of a top level goal. In [18], a backward analysis is presented to infer sufficient groundness condition for error free execution. This is no coincidence since information derived by a forward analysis can be derived by a backward analysis and vice versus [7,19].

*Pos*-based goal-independent groundness analysis enjoys the property of being condensing [16,20,23]. An analysis  $F$  that infers output information  $F(\phi)$  from input information  $\phi$  is condensing if  $F(\phi \sqcap \psi) = F(\phi) \sqcap \psi$  for any  $\phi$  and  $\psi$ . Thus,

a condensing analysis can be performed with partial input information  $\phi$  and its output be conjoined with additional input information  $\psi$  to obtain the output that would result from analyzing the program with complete input information  $\phi \sqcap \psi$ . Condensing has been studied exclusively for goal independent analysis. Condensing can be used to retrieve abstract answers but does not precisely keep track of dependencies between a top level call and a descendant call because the projection operator discards useful information that is essential for maintaining such dependencies.

*Example 4.* Consider the quicksort program in Section 6.2. A non-parametric *Pos*-based goal dependent analysis infers  $call\_app(y_1, y_2, y_3) :- y_1$  from analysis input  $call\_qs(x_1, x_2) :- true$  and it infers  $call\_app(y_1, y_2, y_3) :- y_1 \wedge y_2$  from analysis input  $call\_qs(x_1, x_2) :- x_1$ . The second call pattern  $y_1 \wedge y_2$  for **app**/3 cannot be obtained as the conjunction of the call pattern  $x_1$  for  $qs(x_1, x_2)$  in the second analysis input and the first call pattern  $y_1$  for **app**/3.

## 8 Conclusion

We have proposed an approach to parametrizing a base analysis by lifting its primitive abstract domains to their cardinal powers and obtained a parametric groundness analysis for logic programs using this approach. We have also used positive propositional formulas to encode abstract properties and presented experimental results on a suite of benchmark programs. The experiments show that the parametric groundness analysis is as fast as the non-parametric groundness analysis from which it is obtained.

## References

1. G. Amato and F. Scozzari. Optimality in goal-dependent analysis of sharing. *TPLP*, 9(5):617–689, 2009.
2. T. Armstrong, K. Marriott, P. Schachte, and H. Søndergaard. Two classes of Boolean functions for dependency analysis. *Science of Computer Programming*, 31(1):3–45, 1998.
3. B. Blanchet. Escape analysis for java<sup>tm</sup>: Theory and practice. *ACM Trans. Program. Lang. Syst.*, 25(6):713–775, 2003.
4. R. Chatterjee, B. G. Ryder, and W. A. Landi. Relevant context inference. In *POPL '99: Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 133–146, New York, NY, USA, 1999. ACM.
5. M. Codish and B. Demoen. Analysing logic programs using “Prop”-ositional logic programs and a magic wand. *Journal of Logic Programming*, 25(3):249–274, 1995.
6. A. Cortesi, G. Filé, and W. Winsborough. Optimal groundness analysis using propositional logic. *Journal of Logic Programming*, 27(2):137–168, 1996.
7. P. Cousot. Semantic foundations of program analysis. In S.S. Muchnick and N.D. Jones, editors, *Program Flow Analysis: Theory and Applications*, chapter 10, pages 303–342. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981.
8. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Principles of Programming Languages*, pages 269–282. The ACM Press, 1979.

9. P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(1, 2, 3 and 4):103–179, 1992.
10. P. Cousot and R. Cousot. Abstract interpretation frameworks. *J. Logic and Comput.*, 2(4):511–547, 1992.
11. P.W. Dart. On derived dependencies and connected databases. *Journal of Logic Programming*, 11(2):163–188, 1991.
12. S. K. Debray and R. Ramakrishnan. Abstract interpretation of logic programs using magic transformations. *J. Log. Program.*, 18(2):149–176, 1994.
13. R. Giacobazzi and F. Ranzato. Functional dependencies and moore-set completions of abstract interpretations and semantics. In J. Lloyd, editor, *Proceedings of the 1995 International Symposium on Logic Programming*, pages 321–335. The MIT Press, 1995.
14. R. Giacobazzi and F. Ranzato. The reduced relative power operation on abstract domains. *Theoretical Computer Science*, 216:159–211, 1999.
15. M. Hermenegildo, R. Warren, and S.K. Debray. Global flow analysis as a practical compilation tool. *Journal of Logic Programming*, 13(1, 2, 3 and 4):349–366, 1992.
16. D. Jacobs and A. Langen. Static analysis of logic programs for independent and parallelism. *Journal of Logic Programming*, 13(1–4):291–314, 1992.
17. N. D. Jones and H. Søndergaard. A semantics-based framework for the abstract interpretation of Prolog. In S. Abramsky and C. Hankin, editors, *Abstract Interpretation of Declarative Languages*, pages 123–142. Ellis Horwood Ltd, 1987.
18. A. King and L. Lu. A backward analysis for constraint logic programs. *Theory and Practice of Logic Programming*, 2(4&5):517–547, 2002.
19. A. King and L. Lu. Forward versus backward verification of logic programs. In C. Palamidessi, editor, *Proceedings of Nineteenth International Conference on Logic Programming*, volume 2916 of *Lecture Notes in Computer Science*, pages 315–330, 2003.
20. A. Langen. *Advanced techniques for approximating variable aliasing in logic programs*. PhD thesis, Los Angeles, CA, USA, 1991. Chairman-Jacobs, Dean.
21. D. Liang and M. J. Harrold. Efficient computation of parameterized pointer information for interprocedural analyses. In P. Cousot, editor, *Static Analysis, 8th International Symposium, SAS 2001, Paris, France, July 16-18, 2001, Proceedings*, volume 2126 of *Lecture Notes in Computer Science*, pages 279–298. Springer, 2001.
22. L. Lu. Parameterizing a groundness analysis of logic programs. In P. Cousot, editor, *Proceedings of the Eighth International Static Analysis Symposium*, volume 2126 of *Lecture Notes in Computer Science*, pages 146–164. Springer, 2001.
23. K. Marriott and H. Søndergaard. Precise and efficient groundness analysis for logic programs. *ACM Lett. Program. Lang. Syst.*, 2(1-4):181–196, 1993.
24. F. Scozzari. Logical optimality of groundness analysis. *Theor. Comput. Sci.*, 277(1-2):149–184, 2002.